Aula V

Dip. di Matematica “G. Castelnuovo”

Univ. di Roma “La Sapienza”

Liste

Igor Melatti

Slides disponibili (assieme ad altro materiale) in:

`http://www.dsi.uniroma1.it/~melatti/programmazione1.2007.2008.html`

Creare una lista da una stringa

- Ripasso: struttura di una lista (unidirezionale)

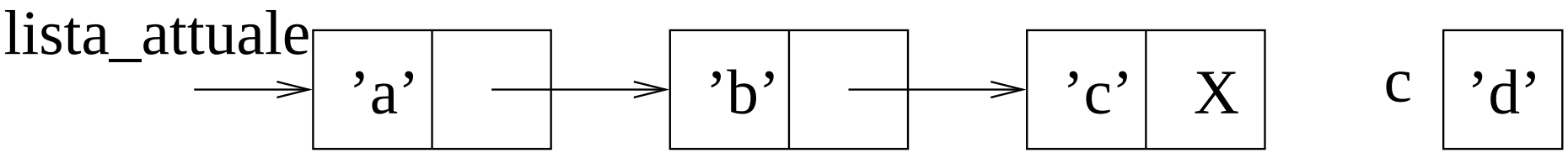
```
typedef struct elemento_di_una_lista {
  char elemento;
  struct elemento_di_una_lista *punt_succ;
} elemento_di_una_lista;

typedef elemento_di_una_lista *lista;
```

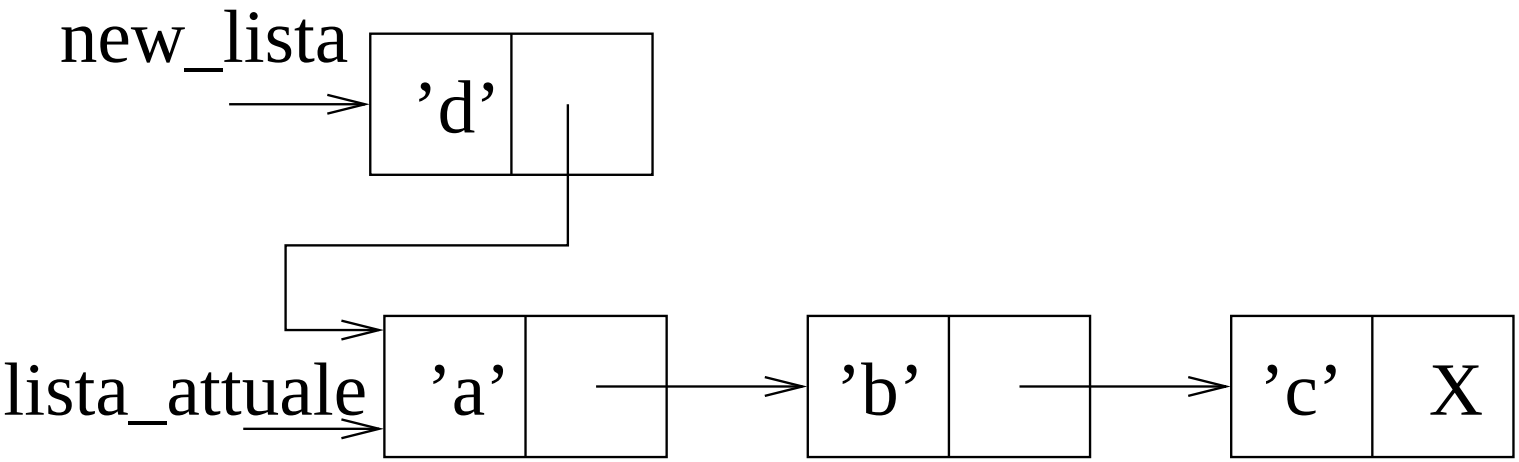
- Prototipo: `lista crea_lista(char *s)`
- Soluzione “pulita”: ho una funzione che realizza l’inserimento
  - data una lista l ed un elemento e, inserisce e in l
  - lo si può fare sia in testa che in coda
  - per come è definita sopra, conviene farlo in testa

Creare una lista da una stringa

```
lista inserisci_in_testa(char c, lista lista_attuale)
{
 lista new_lista = (lista)malloc(sizeof(elemento_di_una_lista));
 new_lista->elemento = c;
 new_lista->punt_succ = lista_attuale;
 return new_lista;
}
```

Creare una lista da una stringa





Creare una lista da una stringa

```
lista crea_lista(char *s)
{
 int i;
 lista L = NULL; /* o anche lista L = (lista)0; */
 for (i = 0; s[i] != '\0'; i++)
  L = inserisci_in_testa(s[i], L);
 return L;
}
```

Creare una lista da una stringa

```
void stampa_lista(lista L)
{
 lista L_punt;
 for (L_punt = L; L_punt != NULL; L_punt = L_punt->punt_succ)
  printf("%c", L_punt->elemento);
 printf("\n");
}

int main()
{
 stampa_lista(crea_lista("ciao"));
}
```

Creare una lista da una stringa

```
void stampa_lista_rec(lista L)
{
 if (L != NULL) {
  printf("%c", L->elemento);
  stampa_lista_rec(L->punt_succ);
 } else
  printf("\n");
}
```

Alcune considerazioni

- C’è una cosa che non va proprio benissimo
- La lista creata resta “appesa”
  - viene allocata (dalla `malloc`)
  - ma non viene deallocata (non c’è la `free`)
- In questo caso, è poco male, il programma termina subito dopo
  - quando un programma termina, tutta la sua memoria allocata viene deallocata automaticamente
- Ma se la lista fosse stata molto lunga e/o con elementi molto grossi
  - ora l’elemento è un `char`, ma niente vieta di metterci strutture molto grosse

Alcune considerazioni

```
typedef struct elemento_di_una_lista {
  double elemento[20000];
  struct elemento_di_una_lista *punt_succ;
} elemento_di_una_lista;

typedef elemento_di_una_lista *lista;
```

- Una lista di 10000 elementi occuperebbe (su una macchina a 32 bit) $10^4 \times (2 \times 10^4 \times 8 + 4)$ bytes, ovvero circa 1.6GB, che è un bel po'
- Dopo che la si è usata, se il programma non finisce subito è meglio liberare la memoria

Alcune considerazioni

- Ad esempio, potrebbe essere richiesto di leggere 10000 elenchi di valori numerici, e poi fare delle statistiche
  - ciascun elenco con 20000 `double`
- Una volta scritte queste statistiche, viene richiesto di leggere dati di diverso tipo ma ugualmente “pesanti”
  - ad esempio altri 10000 elenchi di stringhe
- Se già ho occupato 1.6GB, potrebbe non bastare più la memoria per gli elenchi di stringhe
  - infatti, se non libero con la `free` la memoria che è stata allocata con la `malloc`, quella memoria non può essere usata
- Ma sarebbe cattiva programmazione, perché la memoria c’è , basta liberarla

Alcune considerazioni

```
void distruggi_lista(lista L)
{
 lista L_punt;
 while (L != NULL) {
  L_punt = L;
  L = L->punt_succ;
  free(L_punt);
 }
}
```

- Riscrivere il `main` usando propriamente la `distruggi_lista`
- I listati di cui sopra stampano la stringa data al contrario; correggere affinché ciò non accada
  - o modificando la `crea_lista`
  - o modificando la `stampa_lista_rec` (non facile, ma provateci)
  - o facendo l'inserimento in coda
    - in quest'ultimo caso, provare anche a fare l'inserimento in coda efficiente, ovvero definendo una struttura `lista_con_capo_e_coda` con anche il puntatore all'ultimo elemento (difficile)
- Fare la versione ricorsiva di `distruggi_lista`

Esercizi

- Riscrivere il `crea_lista` in maniera che abbia il seguente prototipo:
  `void crea_lista(lista *, char *)`
  - ovvero, io gli passo un qualcosa di tipo lista e lui me lo *modifica* mettendoci dentro i caratteri della stringa passata come secondo argomento
  - attenzione, lo `*` di `lista *` è importante: perché?
- Scrivere una funzione che cerca un elemento in una lista
- Scrivere una funzione che prende una lista e due elementi, e sostituisce tutte le occorrenze del primo elemento con il secondo

Esercizi

- Scrivere una funzione che prende una lista l e ritorna *un'altra* lista m che è uguale ad l ma rovesciata
- Scrivere una funzione che prende una lista l e ritorna *la stessa* lista rovesciata (difficile)
  - ovvero, la memoria allocata resta sempre la stessa, ma cambiano i puntatori